**Versioning of composed objects in hierarchically composed workspaces**

*Abstract*: The paper presents a study of composed object formal terminology and principles from versioning perspective. The practical aspect of the study includes defining the rules over versioning transaction in hierarchical workspace environment.

**Introduction**

In [4, p.406] says that *“…a modern workspace is created “behind the science” to perform a particular user-selected task…”* In this article we will present some definitions, principles and rules over versioning of composed objects. In next section we … In section 2 we introduce the definition of composed object as well as some conclusion/rules for their versioning. In section 3 we made an attempt to align current research with a previous one and the …

In section 5 we will investigate versioning transaction among two workspaces – ancestor workspace and offspring workspace.

**….**

**Section 2 Related works**

[G98] – some terms

[G172] - terms

**…**

**Section 3 Hierarchy of workspaces and transactions over them**

In [6] we have presented a model of environment with hierarchical composition of workspaces. The presented model



Figure 1 Example of visibility principles and the consequence

stands on the following two principles for object’s version visibility:

**Principle 1:** For each workspace all local versions of versioned objects are visible within the workspace in despite of existence of any other versions in parental workspaces.

**Principle 2:** One version of versioned object from certain workspace is visible in all children workspaces unless there are no local versions (e.g. main principle).

From the both principles we can obtain the following.

**Consequence 1:** Let object Vo1 does not have local version in workspace B2. Then object Vo1 is visible in workspace B2 with its version situated in the closed parental workspace where it has a local version (figure 1).

The above model principles are incorporated with the following two transactions over versioned object: version propagation to parental workspace; version put-back from parental workspace. The version propagation is similar to check-in operation in commonly used version control systems [8]. Nevertheless of the similarity the main difference is that using check-in transaction developer uploads an object version to a common repository. In our model the term workspace is closer to the term of sub-repository and the presented transactions are defined as interaction between two sub-repositories. The essence of propagation transaction is to distribute a version of an object from one workspace to its parent workspace where in combination with Principle 2 we achieve automatic object’s version distribution among sibling workspaces. The opposite transaction of put-back could be described as a decline of an object’s version and its automatic substitution (Consequence 1) with its visible version in the parent workspace.

**…**

**Section 4 Versioning of composed object**

In current section we present some rules for versioning of composed objects of second level. Nevertheless we accent on objects of second level, all deducted rule pretend to be recursively valid also for N level objects. Prefacing the versioning rules lay out, let introduce the following definitions.

In current paper we will define the *composed object* as an object that is the root node of a tree where all nodes represent objects. The *level of a composed object* will be understood as the height of the object’s tree or sub-tree for the respective root-object. Respectively *sub-object* can be regarded as an object that is a node in tree of objects but it is not the root node in the tree. All these definitions in combination of the definition for versioned object […] lead us to the following:

**Definition**: *Composed versioned object* is a versioned object that is the root node of tree where all nodes represent versioned objects.

Having the above definition we can regard all ordinary versioned object can be regarded as composed versioned object of first level.



Figure 1 Class diagram of second level composed object

Upon of a change in the composition relation between versions of two composed objects we should regard them as different versions (Figure 2). When we exclude a sub-object from certain composed object we gain a new version of the composed object without changing the version of the sub-object. The situation is similar as well in the opposite scenario – when we include certain object as a sub-object to certain composed object we gain a change in the version of the composed object without changing the version of newly added sub-object.



Figure 2 Change in the composition structure via new versions of the composed object

Another characteristic of the composed object is the fact that when we change any of its sub-object we gain indirectly also new version of the composed object (Figure 3). In addition new sub-object creation for a composed object could be regarded as a special case of this rule.



Figure Indirect version change of Composed Object via change of the sub-object

In the opposite situation when we change the composed object then we don’t change any of its sub-objects (Figure 4).



Figure Changes in composed object does not have influence on sub-object’s version

As conclusion of the above two facts we could state the following: when we change one sub-object of a composed object, this change does not initiate any change of sibling sub-objects of the changed one.



Figure 5 Change of one sub-object doesn’t change its siblings sub-objects

**Section 5 Composed object versioning in hierarchy of workspaces**

Let have in the ancestor workspace visible version v2 of Object B. Let have a change in composed object B in the offspring workspace that is a new sub-object (Figure 6 - from v2 to v3). When we propagate only the newly created sub-object’s version (green arrow with number 1) then in ancestor workspace we will have two independent objects’ versions – Object B v.2 and Object A v.1. Nevertheless upon subsequent propagate of Object B v.3, it will be published with its composition data (green arrow with number 2). Therefore in the ancestor workspace we will gain automatic change of the composition between Objects B and A (Figure 6 – dotted green line).

Като начало нека разгледаме вариант при който промяната на в една композизия от обекти, представлява създаване на под-обект, (). Публикуването само на версията на под-обекта е възможно и тя не води до някаква промяна във версията на супер-обекта в родителското работно пространство. Въпреки това при последващо публикуване версията на супер-обект в родителското работно пространство ще доведе до автоматична промяна (в рамките на работното пространство) на композиционната схема на обектите ( – зелено пунктираната стрелка). On other hand composed object propagation supposed to be done along with the newly created sub-object (Figure 6 arrows №2).



Figure New sub-object in a composed object

Let have in the ancestor workspace visible version v2 of Object B. Let have a change in composed object B in the offspring workspace that is a new sub-object (Figure 7- from v2 to v3).

При положение, че имаме индиректна промяна на супер-обект – в следствие от нова версия на негов по-обект, следва да отбележим, че е би следвало да може да се публикува самостоятелно новата версия на под-обект в родителското работно пространство. Това ограничение следва от факта, че наличието на нова версия на под-обекта, предполага наличието на нова версия на супер-обекта (Фиг. 32 – червената стрелка с №1). Също така следва да отбележим, че публикуването на новата (индиректно създадена) версия на супер-обекта, трябва да става в комплект с версията на под-обекта, породил индиректната промяна (Фиг. 32 – зелената и жълтата стрелки с №2). Това е породено от факта, че двете версии са взаимосвързани.



Figure Indirect change of composed object caused by change in a sub-object

При промяна на композиция, която представлява отделяне на под-обект от супер-обект, публикуването на супер-обекта в родителското пространство води само до отразяване на промяната в композицията, без да се променя версията на под-обект, който в новата версия на супер-обекта, вече не съставна част от него.



Figure Composition change via change only in composed object

The composed object model and the visibility principles from the previous section lead us to the principle of composed object visibility: For a sub-object version is a part of a super-object version composition, the sub-object version is not visible is a workspace where the super-object version is not visible.

**Conclusion**

The paper presents a study of composed object principles from versioning perspective. The practical aspect of the study includes defining the rules over versioning transaction in hierarchical workspace environment.

**References**

[4]

[98]

[172]

[313]